# MIRTO: an Open-Source Robotic Platform for Education

K. Androutsopoulos L. Aristodemou J. Boender
M. Bottone E. Currie I. El-Aroussi B. Fields
L. Gheri N. Gorogiannis M. Heeney M. Micheletti
M. Loomes M. Margolis M. Petridis A. Piermarteri
G. Primiero F. Raimondi N. Weldin

June 14, 2018

**Abstract**

This paper introduces the MIddlesex RoboTic platfOrm (MIRTO), an open-source platform that has been used for teaching First Year Computer Science students since the academic year 2013/2014, with the aim of providing a physical manifestation of Software Engineering concepts that are often delivered using only abstract or synthetic case studies. In this paper we provide a detailed description of the platform, whose hardware specifications and software libraries are all released open source; we describe a number of teaching usages of the platform, report students' projects, and evaluate some of its aspects in terms of effectiveness, usability, and maintenance.

## 1 Introduction

In 2013 the Department of Computer Science at Middlesex University took the decision of re-designing the first year of the Computer Science degree, with the aim of addressing some of the issues with the previous course format: progressive disengagement, negative feedback about course content in terms of employability and "practical" experience, low attendance rate, etc. Due to the diverse range of academic backgrounds of first year students at Middlesex University, another problem to be addressed was the definition of a course format that could accommodate a non-uniform class of students.

The teaching team decided to adopt a problem-based approach to teaching, with a focus on the so-called inverted curriculum where students learn theory whilst they are doing practical exercises [11] and project-centered delivery, providing detailed material that students could use in workshops under the supervision of members of staff. Assessment is performed on a daily basis through so-called *Student Observable Behaviours* (SOBs), which can be

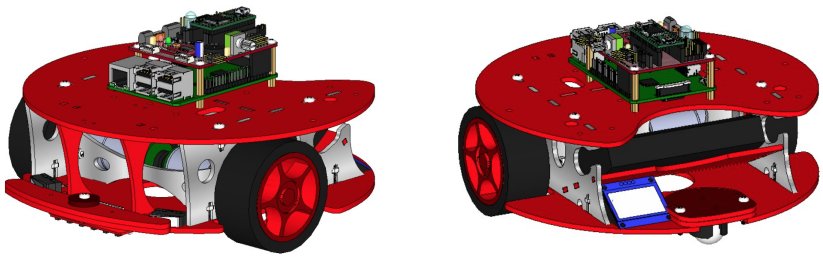

Figure 1: MIRTO fully assembled (front and rear views)

thought of as fine-grained learning outcomes, or *capabilities*, such as "Build and test simple combinatorial logic circuits using at least two different gates in hardware" or "Write a simple recursive function to carry out a well-defined task over lists or integers, test the function and explain how it works". Observation of behaviours is supported by a bespoke assessment tool available at `https://bitbucket.org/mdxmase/sobmonitor`, with the aim of addressing the known limitations of self-paced learning and constructivist approaches [16, 12]: indeed, while students can work in a very flexible way, we are nevertheless able to track their progress both in terms of attendance and progress.

The teaching team decided to place particular emphasis on *physical* manifestations of computing through the use of hardware resources, in an attempt to create a *syntonic* environment (in the sense of [14]), in which students could "*establish a firm connection between personal activity and the creation of formal knowledge*". This approach is particularly useful for code comprehension, but also to cover with *practical and concrete* projects some of the topics typical of Software Engineering, such as agile development in a team, continuous integration, and test-driven development. More specifically, physical computing provides an opportunity for *conceptual blending* [8]: by asking students to work in both abstract and physical spaces, they create blends that enable rich conversations — the behaviour of their code blends with the behaviour of a robot (or device), and the latter is observable in explicit ways.

In this paper we present MIRTO, the MIddlesex RoboTic platfOrm, showing how it can be used to cover several of the Knowledge Areas in the ACM Computer Science curriculum [10], with a particular focus on the Software Engineering knowledge area. In particular, we provide hardware details in Section 2 and software details in Section 3; example applications for teaching and students projects are reported in Section 4, while an evaluation and a comparison with other existing platforms are provided in Section 5. We conclude in Section 6.

## 2 Hardware details

Mirto is a two-wheel robot of circular shape, with a diameter of approximately 20 cm and height 10 cm, see Figure 1. The main components are:

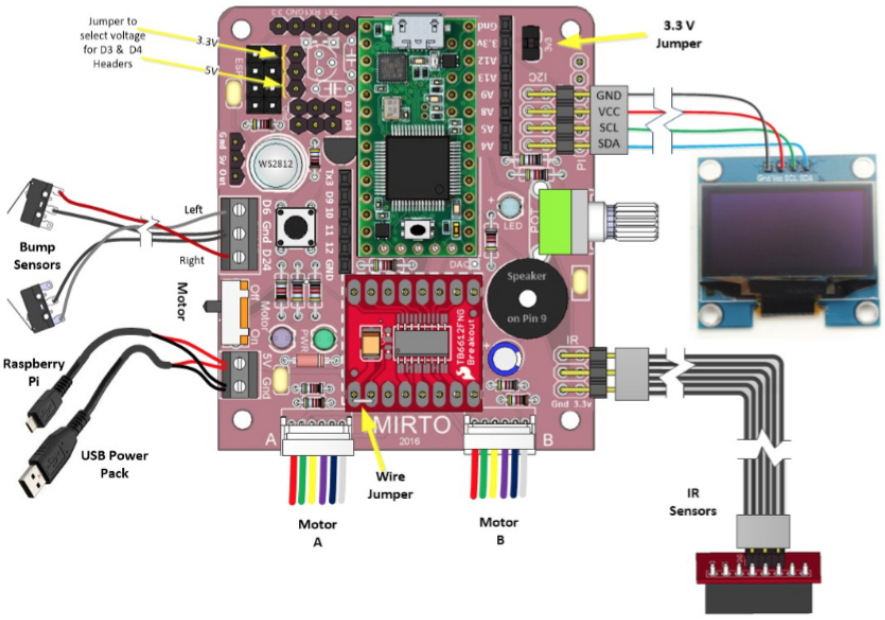


Figure 2: MIRTO PCB

- A pair of 5 V 1:34 geared motors with encoders connected to 1/10 scale car wheels.
- Off-the-shelf components: bump contact sensors, infra-red sensors, potentiometer, digital switch, piezo buzzer, 5-line LCD screen.
- A Teensy 3.2 micro-controller: this is an Arduino-compatible ARM micro-controller running at 72 MHz, 256 Kbytes of flash memory, 64 Kbytes of RAM, 33 usable PINs, USB and serial communication ports.
- The wheels and the off-the-shelf components are connected to the Teensy micro-controller by means of a bespoke printed circuit board (PCB) that has been designed at Middlesex (see Figure 2); the design files of the PCB are released open source (see links below) and several companies are available to print them.
- A Raspberry Pi version 3 with 1.4GHz 64-bit quad-core processor, 1 Gbyte RAM, built-in WiFi, 4 USB ports, HDMI, composite audio output and 40-pin GPIO header. The PCB plugs into the Raspberry Pi GPIO pins and communicates with the Teensy over a serial channel. Software details are reported in the next section.

The frame of the robot is made using 3 mm acrylic sheets cut using a laser cutter, locked using threaded nuts and bolts. The robot used in teaching has a 3 mm aluminium sub-frame cut using a water jet cutter; however the all-acrylic versions have proven sufficiently robust. The design files for the frame in DXF format, the design files for the PCB, a full list of parts, and hardware instructions are available at `https://github.com/michaelmargolis/MirtoDesignFiles`. We refer in particular to `http://goo.gl/k26Rfy` for the actual installation instructions. The main parts are shown in Figure 3.

# 3 Software Details

The core software component (and main difference between MIRTO and other robotic platforms) is a bespoke firmware running on the Teensy that allows for the interaction of the micro-controller with separate clients over a streaming connection (typically a serial connection). The firmware has been developed at Middlesex University and is a service-based abstraction of sensors and actuators for micro-controllers called the *Arduino Service Interface Protocol* (ASIP [2]). The protocol is a bi-directional text-based messaging system. As an example, the following message sent *to* a board running ASIP turns digital PIN 13 to HIGH: `I,D,13,1`, while the following message *from* an ASIP board reports the state of 6 analog pins: `@I,a,6,{0:21,1:0,2:1024,3:789,4:0,5:0}` Similar messages are used to control the robot[1], for instance the message `M,m,0,80` to the board sets the wheel with ID 0 (the left wheel) to 80% power moving forward, and messages from the board report the values of bump sensors, infrared sensors, potentiometers, etc.

We refer to [2] and to the ASIP software repository available at `https://github.com/mdxmase/asip` for additional details on the protocol and for its implementation. The key point of this architecture at the level of the micro-controller is that the robot *does not need to be pre-programmed with a specific firmware for specific tasks*. Instead, it can be controlled by a streaming connection that can be implemented in a number of programming languages. As a result, the students are not bound to a programming language or to a specific IDE such as the Arduino IDE to use the robot. Additionally, as the ASIP communication assumes a stream, several options are possible to connect to the board: directly using the USB port on the Teensy, using the serial pins connected to the GPIO pins of the Raspberry Pi, or over TCP using appropriate bridges TCP/serial, such as the Raspberry Pi itself or a system-on-a-chip (SoC) such as the ESP8266.

We have built client libraries for ASIP in several programming languages: Java (`https://github.com/fraimondi/java-asip/`, C (`https://github.com/fraimondi/c-asip`), Python (`https://github.com/gbarbon/python-asip`), Racket (`https://github.com/fraimondi/racket-asip`), and Erlang (`https:`

[1] Note that the ASIP firmware running on the Teensy could be replaced with pure Arduino code; in this case, which is useful if a real-time controller is needed, the robot can be controlled without a streaming interface.

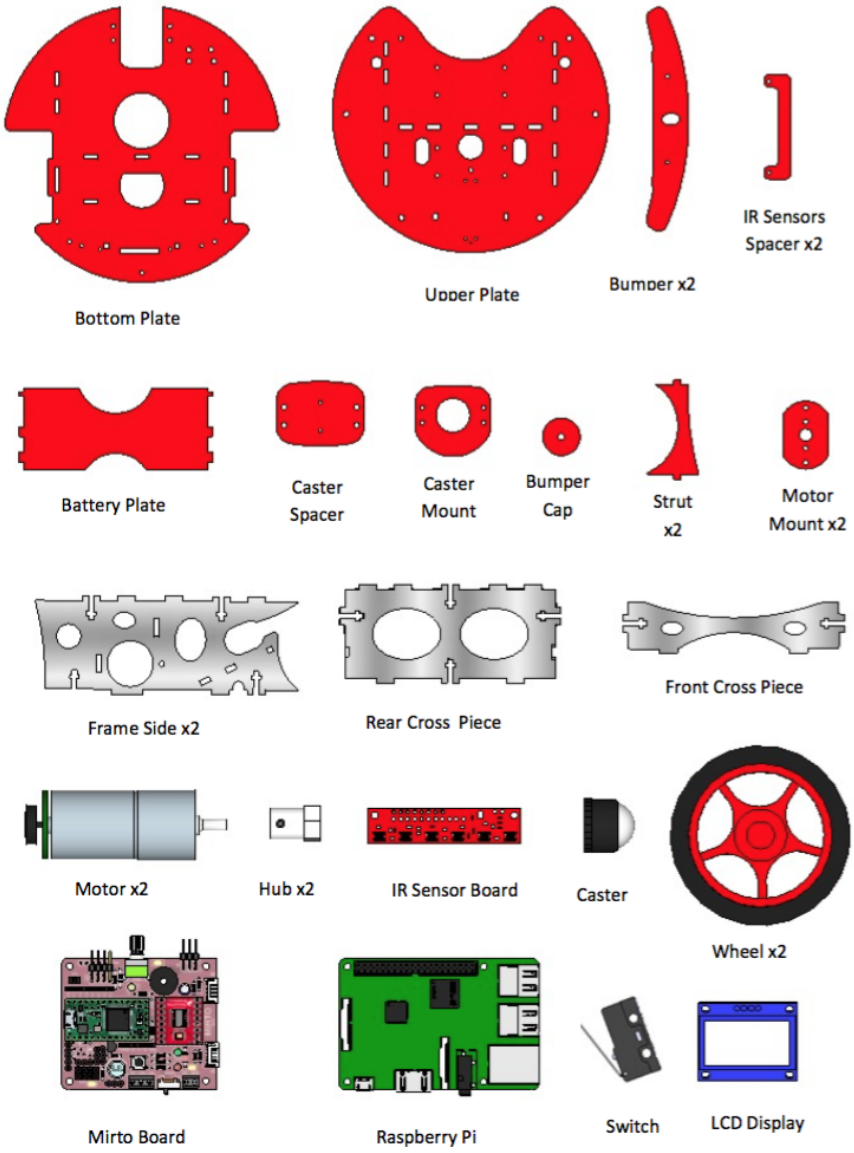


Figure 3: MIRTO main parts

```
(open-asip "COM3")
(setLCDMessage "Hello World" 0)
(setMotors 90 90)
(sleep 1)
(stopMotors)
```

Figure 4: Controlling MIRTO using Racket over serial

//github.com/ngorogiannis/erlang-asip). Note that Erlang is used by students in the second year as part of their module "Distributed Systems and Networking".

Racket [9] is a LISP-like programming language used since 2013 to teach foundations of programming to First Year Computer Science students at Middlesex University. Figure 4 shows an example Racket program that controls a MIRTO robot connected to a Windows computer using a USB cable to the Teensy USB port. The program opens a stream on serial port COM3 (the client can also be run on Linux and Mac machines, changing the name of the serial port), then it writes a message on the LCD screen, then it sets the motors forward at 90% power and finally, after one second, it stops the motors. We refer to [5] for details on how Racket can be used in conjunction with MIRTO to teach functional patterns that have a *physical* manifestation.

In their second and third year, Middlesex Computer Science students use the Java, Erlang, and Python programming languages. Figure 5 shows an example of how the robot can be controlled over TCP. In this case, the Java code runs on a client connected to a network and the robot needs to be connected to the same network. This example assumes that the robot has IP address 192.168.42.1. The Java code creates a connection to this address, then it writes a line to the LCD screen, and finally it moves the robot forward at 90% power for one second. Notice how the code is nearly identical to the code in Figure 4: this allows the students to focus on the *architecture* of the system, which in the case of a networked robot is completely different from a USB connection. There is also a logistical difference; when on a network, the robot could be in a separate room.

Students can also use the Raspberry Pi layer to control the robot. The Raspberry Pi can be configured in two different modes: either as an access point, creating a new wireless network and acting as a DHCP server, or as a client of pre-existing networks. In both cases, students have access using SSH to MIRTO and can deploy code on the Raspberry Pi, which acts as a standard headless server. This enables the construction of complex interactions between systems. For instance, the Raspberry Pi could monitor a Twitter account and react to specific messages. We refer to the next section for details of possible applications.

```
// [...]
JMirtoRobotOverTCP robot =
     new JMirtoRobotOverTCP();
robot.initialize("192.168.42.1");

// Writes a line to LDC
robot.writeLCDLine("Hello World", 0);

// Move forward for 1 second
robot.setMotors(90, 90);
Thread.sleep(1000);
robot.stopMotors();
```

Figure 5: Controlling MIRTO using Java over TCP

# 4 Applications

In this section we first describe possible teaching applications using MIRTO, suitable for students at different levels. Then, we report on projects developed by students in the past four years.

## 4.1 Teaching applications

The robot enables us to cover several knowledge areas of the ACM Computer Science Curriculum [10]. In addition to Software Development Fundamentals such as conditional structures, iterative control structures, recursion, and abstract types such as Stacks and Priority Queues, MIRTO exposes students to a physical manifestation of concepts such as unit testing and programming using APIs. Moreover, MIRTO can be used to cover specific Software Engineering topics, some of which are exemplified in the following subsections.

### 4.1.1 Continuous integration and delivery

The Raspberry Pi layer of MIRTO runs a standard Debian-based Linux distribution. As a result, all the development tools available on a standard Linux distribution are also available on MIRTO. A workflow that can be used by students is the following: students develop code on their machine using an IDE such as Eclipse, NetBeans or IntelliJ. The code is then committed and pushed to a git repository, typically a GitHub or Bitbucket account (both provide student licenses), where students share code in small groups. The key features of continuous integration and delivery can be covered by installing git on the Raspberry Pi and by defining a "bare" git repository[2] on it. Bare repositories are used to set up "push-to-deploy" workflows: on their machines, students need to add a remote endpoint for the robot, using a command similar to the following (this

[2] https://git-scm.com/book/en/v2/Git-on-the-Server-Getting-Git-on-a-Server

could also be added to the IDE): `git remote add production mirto@192.168.1.1:/production`, where 192.168.1.1 is the IP address of the robot, mirto is the username on the Raspberry Pi, and production is the name of the bare repository on the Raspberry Pi. The repository on the Raspberry Pi is configured with a post-receive hook, typically under `/home/mirto/production/.git/hooks/post-receive` and containing the instructions to be executed when new code is pushed to `production`. A typical exercise for this scenario is the following:

> *"Develop a line following algorithm for MIRTO using the infra-red sensors to detect the presence of a black line. Start by using a threshold algorithm (on/off the line).* **While the robot is following the line, change the algorithm to a proportional controller and deploy the new code using a post-receive hook in git***"*.

Students are given a Bash script containing the basic instructions to build the software on the Raspberry Pi, to stop a running process (the line following algorithm in this case), and to re-start the process. Upon completion of this task, students are encouraged to investigate the use of Jenkins[3], an industry-standard automation tool that allows the construction of complex workflows for continuous delivery / continuous deployment.

### 4.1.2 Test-driven development

Students are required to reason about *functional requirements*, describing how *system data is exchanged using UML*. Figure 6 shows an example UML sequence diagram showing how a message travels from the student laptop to the actual wheel through a TCP/Serial bridge on the Raspberry Pi and ASIP messages. Students are requested to draw these diagrams for all components, thus modelling the bi-directional flow of information and reasoning about the overall architecture (see next section).

Students are required to design *tests* to cover individual functions such as detecting contact with a bump sensor using *unit tests*. We ask students to start from test definition and then to move to the actual implementation, thus introducing students to *test-driven development*. Students are requested to design unit tests for all the components modelled with UML diagrams.

In addition to unit tests, students can also design *integration tests* for all the interfaces that are present, covering the messages exchanged in the UML diagrams. Following integration testing, students can design *system tests* for specific applications, such as line following. Given the black-box nature of system tests, when working in groups students can test other groups' systems and thus also appreciate with concrete instances the notion of *acceptance testing*, thus covering all the steps: unit testing, integration testing, system testing and acceptance testing.

[3] `https://jenkins.io`.

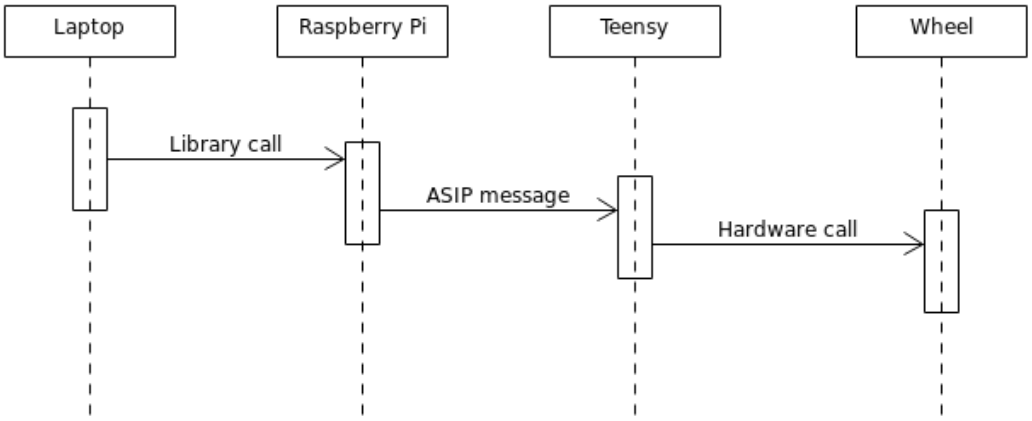


Figure 6: UML sequence diagrams showing data exchanges for moving a wheel (the Raspberry Pi acts as a TCP/Serial bridge).

#### 4.1.3 Reasoning about system architectures

The robot is inherently more complex than software-only systems that can be used in a teaching environment. At the very least, the robot is composed of multiple and independent sensors and actuators that are coordinated by a micro-controller. Students are requested to draw component diagrams of all the possible architectures that can be used with the robot. For instance, Figure 7 shows the Component diagram of the robot in the case in which the micro-controller is connected directly to a laptop using a USB cable (no Raspberry Pi in this case).

Students need to build similar diagrams for the cases in which a Raspberry Pi is introduced. There are two options in this case: in the first option, the code can run on the Raspberry Pi itself (all the client libraries are supported, including Racket). In the second option, the Raspberry Pi can act as a bridge using a TCP connection with an external computer. In this case, the code is running on the laptop and the Raspberry Pi forwards messages in either direction.

All these options allow students to reason about *configuration and release management* for the several projects that can be implemented. Some of these projects include:

- Line following robot: in this case performance is a concern, and therefore the students can choose to run the code on the Raspberry Pi or even directly on the micro-controller.

- Web-based control: in this case the robot becomes a web server and clients can connect to it using a browser. For this architecture, the web applica-

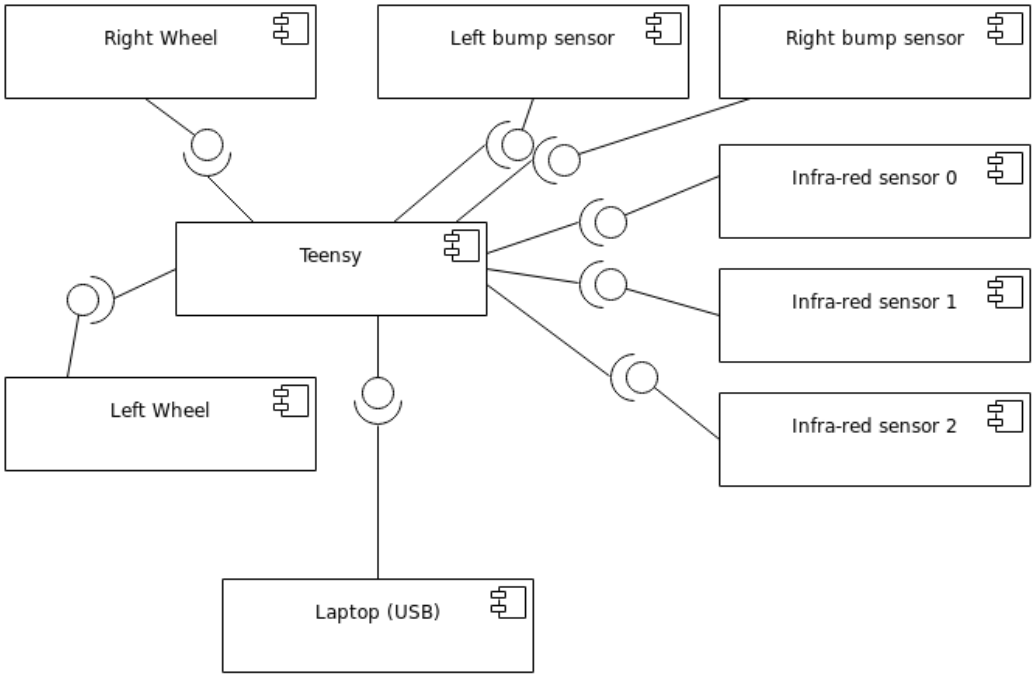


Figure 7: UML Component diagram for a USB-only connection (laptop at the bottom, micro-controller in the middle).

tion needs to run on the Raspberry Pi and it must implement one of the client libraries. First year students typically implement this task using Racket and its built-in web server to drive a robot using key presses on a browser.

- Swarm/multi-robot operations: in this case messaging is typically managed externally to the robot, and therefore the Raspberry Pi becomes a bridge of ASIP messages (see below for a student project in this configuration).

## 4.2 Student projects

We encourage students to work autonomously at projects of their choice throughout the course of their studies, encouraging them to build a *portfolio of projects* using a code repository such as GitHub and Bitbucket. We allocate teaching time for these activities, in particular: in the last four teaching weeks of the First Year, students have the option of working at the "First Year Challenge in Computer Science". This is a challenge funded by an external industrial sponsor that provides a prize of 500 GBP for the "best" student project. For this challenge, students typically work in groups and projects are assessed for their originality, for the Software Engineering approaches used, and for the complexity of the tasks. Assessment is performed by academics in workshops where students can present they work as part of a group presentation (each academic assesses, on average, 5 projects). We report below two submissions received in previous years and a third year project that led to a publication.

### 4.2.1 Voice recognition.

This project by a first year student involved the use of Carnegie Mellon PocketSphinx[4], a lightweight continuous speech recognition engine written in C. The student compiled the code on the Raspberry Pi and modified it so that it could output only specific command such as "move forward". The commands were then streamed to a Racket application that moved the robot accordingly. The only additional component required for this project was a USB microphone connected to the Raspberry Pi.

### 4.2.2 Using Twitter and Image Capturing.

This project by a group of first year students involved the use of the Twitter streaming API[5]. By connecting MIRTO to the internet, students were able to filter tweets containing the keyword `MIRTOBOT`. Then, they could parse specific messages such as movement instructions, but also instructions to take pictures using a USB webcam added to the Raspberry Pi. All the images captured in this way have been made available through a web server running on the robot and written in Racket.

[4] `https://github.com/cmusphinx/pocketsphinx`
[5] `https://developer.twitter.com/en/docs/tweets/filter-realtime/overview`

#### 4.2.3 Swarm robotics

This is a final year project developed as a research investigation into decision making in a multi-robot environment [3]. The student first developed and simulated algorithms for decision making using NetLogo [17]. These algorithms have then been implemented in Java and deployed on Mirto. A short video of is available at `https://goo.gl/D3dUkk`. This project shows that MIRTO can be used not only for teaching, but also to introduce students to research activities.

# 5 Evaluation

In this section we evaluate the MIRTO platform and compare it to other existing solutions. We also discuss our experience since 2013 and report lessons learnt that, we hope, may be useful to other educators who plan to use a robotic platform in their classes.

## 5.1 Qualitative and Quantitative Considerations

In terms of a qualitative financial evaluation, the overall cost per robot *for parts only* is approximately 100 GBP; the most expensive components are the Raspberry Pi (approx 30 GBP) and the Teensy 3.2 (approx 20 GBP). These costs do not include the time required to assemble the robots (approx 1 hour for each robot for an experienced assembler) and to maintain them. We discuss these indirect costs below.

Given that the robots are pre-assembled and that students are provided with high-level libraries and detailed handouts, we have observed that students are able to start working at specific tasks such as "move forward for 1 second" in less than 30 minutes when they are first exposed to the platform. The libraries that we provide allow students to focus on the actual code implementing specific tasks, rather than on the hardware details of the robot. The presence of an LCD screen enables students to connect to the robot easily through a network connection. As a result, from a qualitative point of view we consider the robot *usable*, in that it enables students to reason about their coding or modelling tasks in a relatively short amount of time. The robots are equipped with a 10,000 mAh battery, which allows a full day of teaching (8 hours over multiple sessions). The most energy-hungry task is wheel movement at maximum torque (i.e., uphill or on a "soft" surface).

For a quantitative evaluation, we have measured when a basic SOB associated with the robot was observed, and compared it to all the other threshold SOBs. In particular, after introducing the material for a SOB, we have observed how many students were observed on subsequent days, for a total of 144 students in the academic year 2017/18. Overall, SOBs are observed on average 53 days after the introduction of the corresponding material; the SOB on robots is observed, on average, 26.5 days after the introduction of the material. Only 4 other SOBs are observed (on average) earlier than the SOB on the robot:

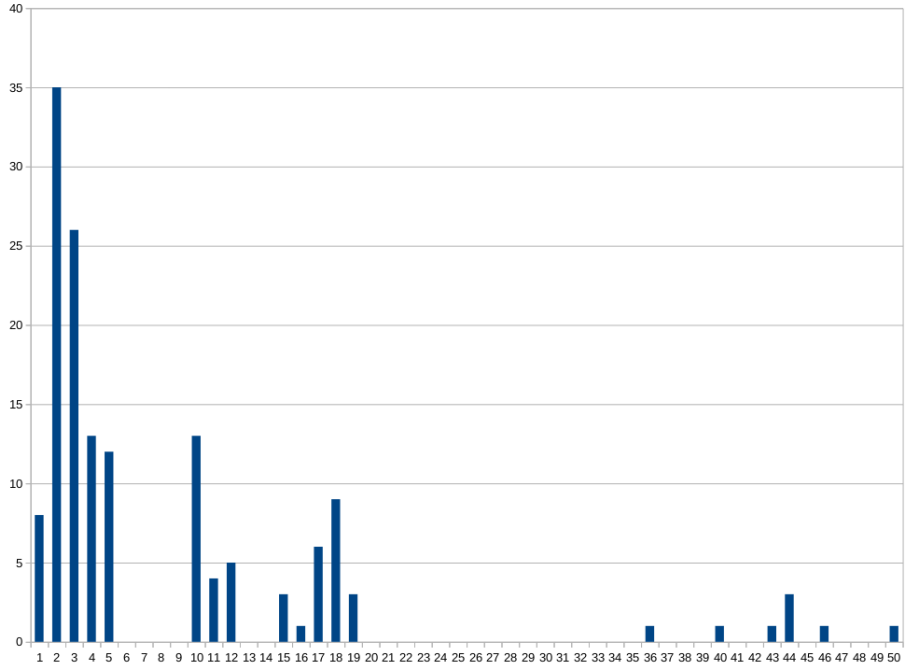


Figure 8: Students observed in each day following the introduction of the material.

2 of them are related to an in-class programming test which is normally observed within a week, as students are required to take the test in specific dates. Interestingly, one of the other two SOBs is related to another exercises with hardware ("Program an array of data to several different data addresses with physical memory"). Figure 8 shows in detail how many students were observed as a function of the number of days after the introduction of the material.

## 5.2 Comparison with other platforms

The number of robotic platforms used in teaching is vast, and a detailed review of the options is beyond the scope of this article. A number of institutions include the actual design and construction of a robot in their teaching material: we have chosen, instead, to provide students with a pre-built platform that can nevertheless be expanded in a number of ways. Our choice has been motivated by the aim of focusing on learning outcomes that are more aligned with a traditional CS curriculum [10]; additionally, as noted by other authors [7], issues associated with robot construction may actually hinder the learning process.

A number of options are available for younger children: several variants of the Bee-Bot platform are commonly used in primary schools, while the Dash and Dot robots by `https://www.makewonder.com/` have a dedicated program-

ming language based on blocks. Micro-controllers are also commonly found in classrooms, including Arduino boards and the UK-specific BBC micro:bit (`http://microbit.org/`) targeted at Year-7 students and including both block-based and textual coding environments.

In higher education, the Lego EV3 platform is probably the most common off-the-shelf platform, see for instance [6]. This platform can be used either as a micro-controller based solution, or in conjunction with a Raspberry Pi through a BrickPi kit (`https://www.dexterindustries.com/brickpi/`). The main difference with respect to our solution is that we release both *hardware and software* as open source. The advantage of open source hardware and software over packaged educational offers is that the robot can be built to address many different educational goals. Perhaps most obvious is that, unlike the Lego EV3 and many other robots designed for education, it is built of standard parts rather than fixed or snapped together components. This enables educators and students to tailor the hardware and software to meet specific educational goals, and in addition to have access to the extensive range of sensor and actuator libraries for the Arduino ecosystem.

In general, the adoption of robots in the classroom is by no means new, see for instance [4] for a systematic literature review and for references to options beyond Lego EV3.

It is normally accepted that robots provide motivational platforms for students to engage with the teaching material; for instance, the work in [13] found a positive correlation between motivation and robots usage using surveys. However, not all interventions with robots are successful in terms of learning; see for instance Fagin [7]: the authors observed that "Students in robotics sections must run and debug their programs on robots during assigned lab times", a time-consuming task that prevents students from focussing on the foundational issues of coding. We argue that, in the case of MIRTO, the provision of high-level libraries in Java and other programming languages enables students to by-pass low-level technical issues.

### 5.3 Lessons learnt

The first version of the robot was designed in 2013 for the academic year 2013/14. It consisted of a Raspberry Pi version 2, an Arduino Uno, and a pair of Hub-ee wheels[6], see Figure 9.

The main focus of the original build was the support of the Racket programming language, as described in [1]. This version of the robot was successful with students, but it suffered from a number of issues:

- The wiring was complex, using standard Arduino pins: the connections were subject to wires being pulled or mis-placed, and therefore the platform required regular maintenance.

[6] `http://www.creative-robotics.com/About-HUBee-Wheels`

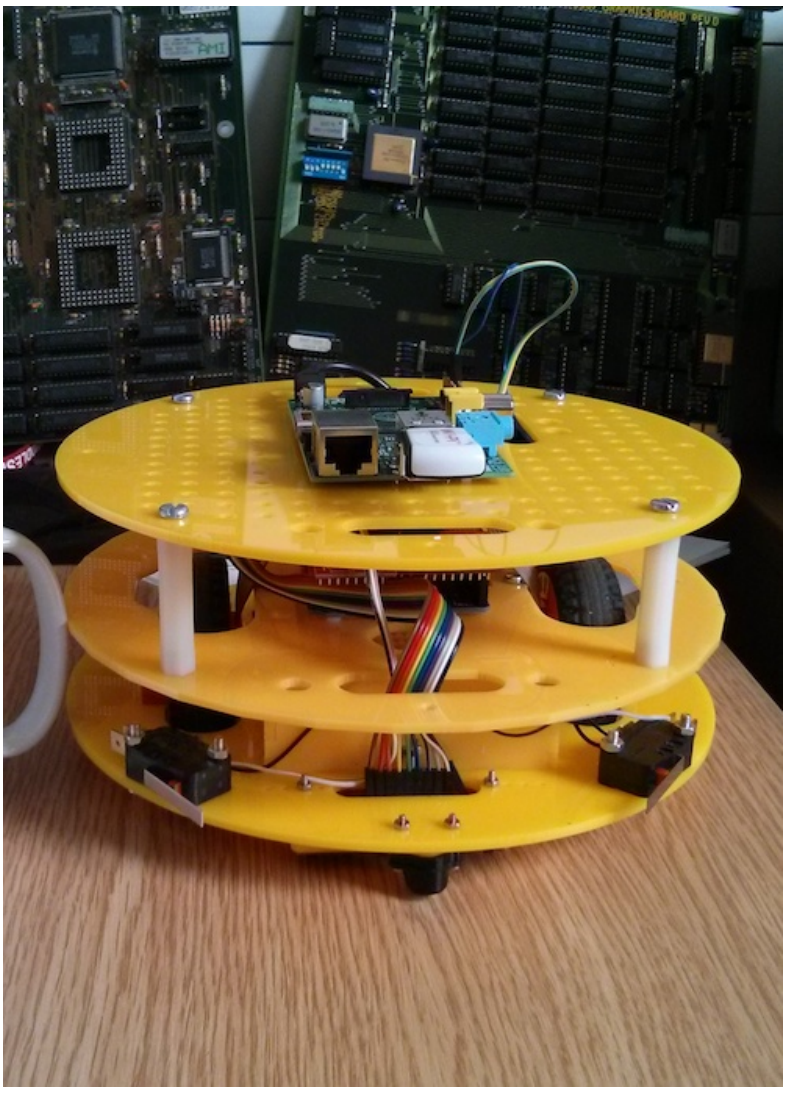

Figure 9: MIRTO Robot: 2014 version

- The structure was made of three layers of acrylic joined by plastic spacers, subject to vibrations and to screws becoming loose.
- The Arduino layer employed Firmata, a 7-bit protocol designed to control Arduino Input/Output pins. While this allowed us to have a functional software prototype in a short time, it also proved extremely difficult to extend with additional features.

Since 2013/14, the robotic platform has been used every academic year for a total of 890 students. Over the years, we have made a number of changes that allowed us to:

- **Simplify assembly and reduce maintenance**: starting in 2015, we have designed our own printed circuit board, resulting in a very small number of wires required. While the design of the circuit board may require an initial time investment, it also reduces the assembly time substantially and, moreover, it simplifies maintenance and reduces the risk of wires being pulled. We have also removed one layer and introduced in 2017/18 a metal sub-frame to make the structure more rigid. The Hub-ee wheels have been replaced with a different (and much cheaper) motor that has proven more robust.
- **Enable hardware extensions**: we have replaced the original Arduino Uno with a Teensy that, at the same price, provides a much greater number of PINs, more RAM, and faster processor. We also moved to a Raspberry Pi version 3, with built-in WiFi and more powerful processor. This has allowed students to connect additional sensors, such as sonar distance sensors and servo motors.
- **Enable software extensions**: we have replaced the Firmata protocol with ASIP [2], a plain-text protocol for the composition of services on the micro-controller. This new protocol has allowed students to design new services and it has also resulted in a research middleware for wireless sensor networks [3].

The staff resources required to employ robots in the classroom should not be underestimated. There is a total of approximately 180 students in the First Year of Computer Science; students are split in groups of approximately 30 students and therefore the same lab sessions are repeated six times every week. For the robotic sessions we normally employ between 7 and 10 robots per class, corresponding to approximately 3 students per robot. We currently have 20 identical robots available that, when required, enable us to cover two parallel sessions. The team involved in teaching the first year includes academic and teaching assistants delivering sessions; teaching assistants are also employed to assemble the robot at the beginning of term (each robot requires approximately 1 hour to be assembled). For the assembly task the department has also recruited Engineering students, providing them with an opportunity to learn relevant skills. A separate group of teaching assistants is in charge of testing, quality

control and software set-up. The ASIP firmware is flashed in a few seconds on the Teensy layer, while the software for the Raspberry Pi is installed by cloning a master SD card prepared by academics. We have an SD card duplicator that can prepare 20 SD cards in approximately 2 hours.

The operations mentioned above are normally performed during non-teaching weeks at the beginning of term. During the teaching term, the robots are kept in a locked room where academics can collect and drop them. Two teaching assistants are in charge of performing a daily check of the robots, making sure that no parts are broken or loose; the same teaching assistants are also responsible for charging the batteries. These activities take approximately 2 hours per day.

To amortise these costs, we have designed the platform in such a way that it can support multiple programming languages, and thus be used not only by first year students, but also by second and third year students both in scheduled classes and for personal projects. The robots are also used for outreach activities with children using Scratch [15]: a Scratch-MIRTO bridge is available at `https://github.com/fraimondi/java-asip`. For Scratch activities, the robot creates a wireless access point and children can employ blocks that we have designed to perform simple tasks, such as moving inside an area delimited by black tape to simulate a Roomba robot: an example program for this task is reported in Figure 10. In this program, the robot starts at speed 100 for both wheels. If it detects a black line (value of infra-red sensor greater than 400), it stops, backtracks with speed -80 for 0.5 seconds, stops again, rotates with speed -80 left and 80 right for 0.5 seconds, and finally restarts again.

# 6 Conclusion

In this paper we have presented MIRTO, an open source robotic platform employed at Middlesex University to teach Software Engineering knowledge areas from the ACM curriculum [10]. We have described both the hardware and the software architecture of the robot and we have provided example activities for the classroom and reported projects developed by students in the past 5 academic years.

An quantitative measure of engagement with the material performed in 2017/18 shows that students tend to engage much earlier with material associated with the robot (and with other "practical" tasks), compared to tasks associated to more theoretical concepts.

We have discussed resource requirements for the deployment of robots in the classroom, both in terms of costs of equipment and in terms of additional man power required for set-up and maintenance.

In addition to its usage in the classroom, we have used MIRTO for outreach activities with audiences ranging from primary school children to science festivals with adults. Given the level of maturity and stability of the platform, we are now beginning to start using the platform for research purposes to study energy consumption in wireless sensor networks and genetic algorithms for strategy

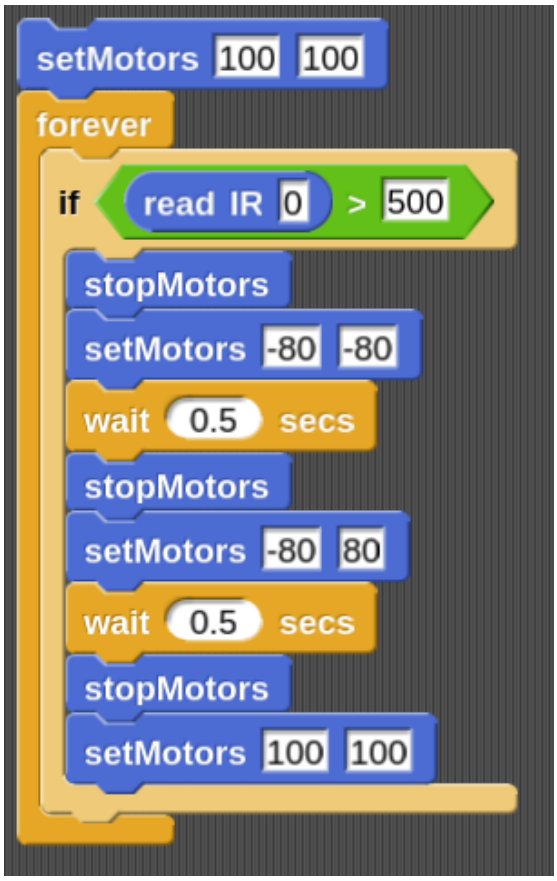


Figure 10: A Scratch program to control the robot

selection in a multi-robot environment.

All the material has been released open source and is available at the following links:

- Design files and assembly instructions:
  `https://github.com/michaelmargolis/MirtoDesignFiles`.
- ASIP firmware for the Teensy layer:
  `https://github.com/mdxmase/asip`
- Java ASIP client library:
  `https://github.com/fraimondi/java-asip`
- C ASIP client library:
  `https://github.com/fraimondi/c-asip`
- Racket ASIP client library:
  `https://github.com/fraimondi/racket-asip`
- Python ASIP client library:
  `https://github.com/gbarbon/python-asip`
- Erlang ASIP client library:
  `https://github.com/ngorogiannis/erlang-asip`

# References


[1] K. Androutsopoulos, N. Gorogiannis, M. Loomes, M. Margolis, G. Primiero, F. Raimondi, P. Varsani, N. Weldin, and A. Zivanovic. A racket-based robot to teach first-year computer science. In *Proceedings of the 7th European Lisp Symposium*, pages 54–62, 2014.

[2] Gianluca Barbon, Michael Margolis, Filippo Palumbo, Franco Raimondi, and Nick Weldin. Taking arduino to the internet of things: The asip programming model. *Computer Communications*, 89-90:128 – 140, 2016. Internet of Things: Research challenges and Solutions.

[3] Luca Battistelli and Giuseppe Primiero. Weighted collective decision making for binary properties in an autonomous multi-robots system. 2018. Under Review.

[4] Fabiane Barreto Vavassori Benitti. Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education*, 58(3):978 – 988, 2012.

[5] Jaap Boender, E. Currie, M. Loomes, Giuseppe Primiero, and Franco Raimondi. Teaching functional patterns through robotic applications. In Johan Jeuring and Jay McCarthy, editors, *Proceedings of the 4th and 5th International Workshop on Trends in Functional Programming in Education,*

*TFPIE 2016, Sophia-Antipolis, France and University of Maryland College Park, USA, 2nd June 2015 and 7th June 2016.*, volume 230 of *EPTCS*, pages 17–29, 2016.

[6] A. Cruz-Martin, J.A. Fernandez-Madrigal, C. Galindo, J. Gonzalez-Jimenez, C. Stockmans-Daou, and J.L. Blanco-Claraco. A lego mindstorms nxt approach for teaching at data acquisition, control systems engineering and real-time systems undergraduate courses. *Computers & Education*, 59(3):974 – 988, 2012.

[7] Barry Fagin and Laurence Merkle. Measuring the effectiveness of robots in teaching computer science. *SIGCSE Bull.*, 35(1):307–311, January 2003.

[8] G. Fauconnier and M. Turner. *The Way We Think: Conceptual Blending and the Mind's Hidden Complexities*. Basic Books, 2002.

[9] Matthew Flatt and PLT. Reference: Racket. Technical Report PLT-TR-2010-1, PLT Design Inc., 2010. `https://racket-lang.org/`.

[10] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, New York, NY, USA, 2013. 999133.

[11] Alison King. From sage on the stage to guide on the side. *College teaching*, 41(1):30–35, 1993.

[12] Paul A. Kirschner, John Sweller, and Richard E. Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, 41(2):75–86, 2006.

[13] Monica M. McGill. Learning to program with personal robots: Influences on student motivation. *Trans. Comput. Educ.*, 12(1):4:1–4:32, March 2012.

[14] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA, 1980.

[15] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for all. *Commun. ACM*, 52(11):60–67, November 2009.

[16] Jonathan G. Tullis and Aaron S. Benjamin. On the effectiveness of self-paced learning. *Journal of Memory and Language*, 64(2):109–118, 2 2011.

[17] Uri Wilensky. Netlogo. Technical report, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999.